Développement Logiciel, 2014

Projet - version simple

Sujet du projet : Vous devez concevoir une suite de programmes pour faire tourner un jeu de labyrinthe, où l'objectif des personnages est de sortir du labyrinthe. Votre but est de créer des NPC (Non-Player Characters, ou personnages non-joueurs) pour peupler le monde du jeu, ainsi que des personnages contrôlés par l'utilisateur.

Cette suite est composée de trois programmes, dont nous décrivons pour chacun les caractéristiques dans la version initiale (a) et la version avancée (b) :

1. **L'atelier** : permet la conception des types de personnages pour notre jeu :
   (a) Définition de l'aspect visuel et des propriétés des personnages,
   (b) Définition des compétences, des comportements v.à.v. des autres personnages, des stratégies pour organiser leur mouvements et leur actions dans le labyrinthe.
2. **L'éditeur de labyrinthe** : permet la conception d'un labyrinthe (grille à deux dimensions de salles) :
   (a) Choix de l'aspect visuel du labyrinthe, de la position des objets et des caractéristiques du terrain de chaque salle du labyrinthe.
   (b) Changement des propriétés de salles qui affectent la navigation des personnages.
3. **Le simulateur de jeu** : simule l'avancement des personnages dans le labyrinthe jusqu'à leur sortie :
   (a) Simulation naïve où les salles et personnages ont des caractéristiques similaires, avec des stratégies de navigation simples,
   (b) Prise en compte des obstacles et autres contraintes spécifiées par le labyrinthe, ainsi que le mouvement et les compétences des personnages, et enfin de l'interaction directe sur le jeu (IHM).

Informations générales

Pensez à regarder régulièrement le site `http://www.lri.fr/~anab/teaching/DevLog/` *pour d'éventuels compléments d'information et documents utiles à la réalisation du projet pourraient être déposés. Ce projet s'étend sur les six dernières semaines d'enseignement (jusqu'à la dernière séance de TP).*

On vous demande de mettre en œuvre les différents concepts et outils vus en cours. En particulier, les trois programmes disposeront des éléments suivants :

1. Une interface graphique (fenêtre et dessin) ;
2. Un système de sauvegarde et chargement ;
3. Une utilisation des threads pour la simulation (chaque personnage devra être géré dans un thread indépendant)

De plus, d'autres éléments vus dans le cours de Développement Logiciel (ou dans vos autres cours portant sur Java) devront être mis en œuvre pour garantir la lisibilité de votre code : utilisation de concepts d'objets, des exceptions, commentaires, etc...

Evaluation

Le projet se découpe en six parties : les trois premières sont consacrées à l'élaboration des versions initiales de chacun des programmes, les trois dernières concernent les versions avancées. Chaque moitié du projet (initial/avancé) est notée sur 13 (26/25 au total, 1 point bonus).

Les critères d'évaluation sont les suivants :
- qualité du code (organisation et structure, clarté),
- mise en œuvre des concepts vus en cours,
- qualité des soutenances (deux au totale),
- qualité du rapport et de la méthodologie (voir les cours de *Génie Logiciel*),
- originalité.

Des malus seront attribués en cas de retard, de non fonctionnement du code, d'absence de rapport...

L'évaluation proprement dite aura lieu lors de deux soutenances avec votre chargé de TP :
1. La **première soutenance** sur les versions initiales aura lieu lors de la **3e séance** de TP-projet : **28 mars**
   - Présentation des versions initiales des programmes.
   - Durée de **15 minutes** maximum.
   - Rendu du **rapport imprimé** de 5 pages maximum comportant :
     - une brève introduction des fonctionnalités de chaque programme,
     - des copies d'écran illustrant le fonctionnement de chaque programme,
     - un bref résumé des fonctionnalités envisagées pour la version avancée,
     - ce que chaque partie du binôme a programmé,
     - un bref résumé des bonnes pratiques de programmation mises en œuvre.
   - Remise des **sources** de votre code **sans bugs** sous un fichier ZIP ou TGZ par mail.
2. La **seconde soutenance** sur les versions avancées aura lieu lors de la **dernière séance** de TP : **18 avril**
   - Présentation des versions avancées des programmes.
   - Durée de **15 minutes** maximum.
   - Rendu du **rapport imprimé** de 5 pages maximum comportant :
     - une brève introduction des fonctionnalités de chaque programme,
     - des copies d'écran illustrant le fonctionnement de chaque programme,
     - un bref résumé des difficultés rencontrées et des perspectives pouvant être envisagées,
     - ce que chaque partie du binôme a programmé,
     - un bref résumé des bonnes pratiques de programmation mises en œuvre,
     - et en plus vos observations sur les paramètres du labyrinthe.
   - Remise des **sources** de votre code **sans bugs** sous un fichier ZIP ou TGZ par mail.

Un point **crucial** est qu'il est **absolument** nécessaire de fournir un programme fonctionnel lors de la soutenance, c'est à dire **sans bugs**. Pensez bien à tester votre programme sur différentes machines pour assurer sa portabilité. Il est plus important de garantir que votre programme fonctionne plutôt que d'ajouter une option sans la tester. Les sources de votre programme doivent être envoyées à votre chargé de TP, **en mettant votre chargé de cours en copie**.

Par ailleurs, il est primordial d'arriver à la soutenance avec un **rapport imprimé** (il peut être en noir et blanc) comportant vos noms, prénoms, l'UE et la date.

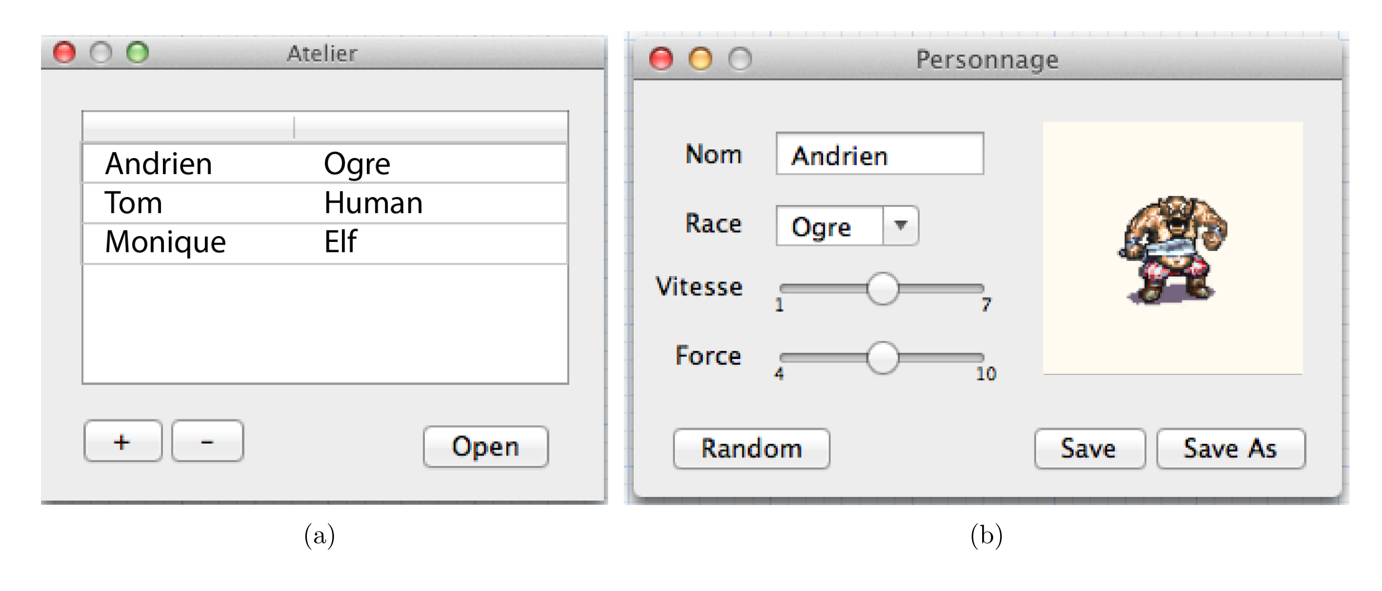
Finalement, vous devez traiter les 2 présentations avec la même attention que l'examen. Vous devez avoir votre rapport avec vous, être préparé pour votre présentation, et être présent à l'heure.

Atelier
Version initiale à rendre pendant la 3e séance de TP-projet

L'atelier permet à l'utilisateur de :
- charger la description (propriétés) d'un personnage existant,
- créer un nouveau personnage, éventuellement en modifiant les propriétés d'un personnage existant.

Fonctionnalités souhaitées :
- Création, sauvegarde et chargement personnage.
- Création par l'utilisateur, aléatoire, ou en modifiant un personnage existant ("save as").

Ressources :
- Si vous les voulez, vous trouverez sur le site du cours une archive contenant des fichiers images des personnages.



(a) (b)

FIGURE 1 – Atelier. Exemples de fenêtres de (a) navigation pour sélection, options pour créer et supprimer des personnages, et de (b) génération de nouveaux personnages (version initiale).

Objectif : Permettre la création d'un personnage à partir de zéro, aléatoirement ou à partir d'une autre.

Chaque personnage est défini par :
- son nom,
- sa race : ogre, humain, ou elfe,
- sa vitesse de mouvement : un entier entre 1-7 pour les ogres, entre 4-10 pour les humains, et 8-10 pour les elfes,
- sa force : un entier entre 4-10 pour les ogres, entre 1-7 pour les humains, et 1-3 pour les elfes,
- et une représentation visuelle, par exemple une image ou une forme géométrique.

Éditeur de labyrinthe
Version initiale à rendre pendant la 3e séance de TP-projet

L'éditeur de labyrinthe permet à l'utilisateur de :
- charger la description (propriétés) d'un labyrinthe existant,
- créer un nouveau labyrinthe en ajoutant les salles (e.x. Sortir),
- ajouter une option permettant de « bloquer » quelques salles pour une certaine période.

Fonctionnalités souhaitées :
- Sauvegarde et chargement de labyrinthe,
- Possibilité de spécifier si une salle peut devenir bloquée pour une période aléatoire (donc on ne peut pas y accéder, et si un personnage est dedans il y est immobilisé).
- Définir la sortie du labyrinthe. Plus précisément il s'agit de choisir la **salle** à partir de laquelle on peut sortir du labyrinthe, qui n'est pas nécessairement sur un bord (ça peut être une salle avec une échelle). Atteindre la sortie reviendra à atteindre cette salle.

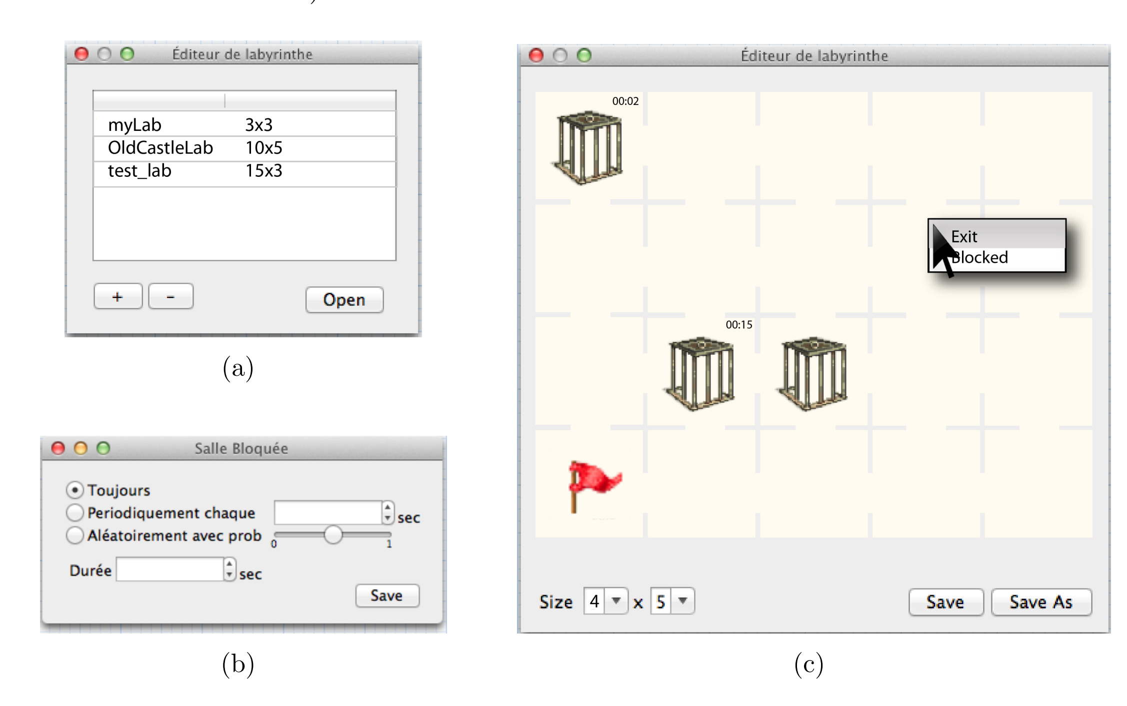


FIGURE 2 – Editeur. Exemples de fenêtres pour (a) ouvrir, créer et supprimer un labyrinthe, (c) pour ajouter les salles, et (b) pour éditer les paramètres de blocage (version initiale).

Objectif :
- Permettre la création d'un labyrinthe N*M, de sa sortie, et de salles bloquées.
- un labyrinthe est représenté par une grille de N*M salles,
- les salles sont soit des salles normales, soit bloquées.
- chaque salle a une représentation graphique,
- les salles bloquées :
  - ne peuvent être traverser pour une certaine période *BlockTime* (qui peut être aussi la durée de la simulation) défini par l'utilisateur.
  - si le blocage n'est pas permanent, c'est soit un blocage périodique qui arrive chaque *BlockPeriod*), soit aléatoire avec une probabilité *ProbBlock* entre 0-1 (tous les deux définis par l'utilisateur).
  - un personnage qui est dans une salle qui devient bloquée doit attendre qu'elle se débloque pour en sortir.

Simulateur
Version initiale à rendre pendant la 3e séance de TP-projet

Le simulateur doit permettre à l'utilisateur de visualiser les positions des personnages et leur déplacements dans le labyrinthe.

Fonctionnalités souhaitées :

- Chargement de personnages créés et sauvegardés en fichiers dans l'atelier,
- Chargement d'un labyrinthe à partir du fichier créé dans l'éditeur de labyrinthe,
- Choix des personnages à mettre dans les salles spécifiques ou aléatoires,
- Visualisation du labyrinthe (salles et déplacements),
- Sauvegarde de tous les déplacements de tous les personnages dans un fichier de log pour trouver les erreurs de programmation plus facilement. Sauvgarder aussi des événements importants (positions initiales, blocage, etc.).

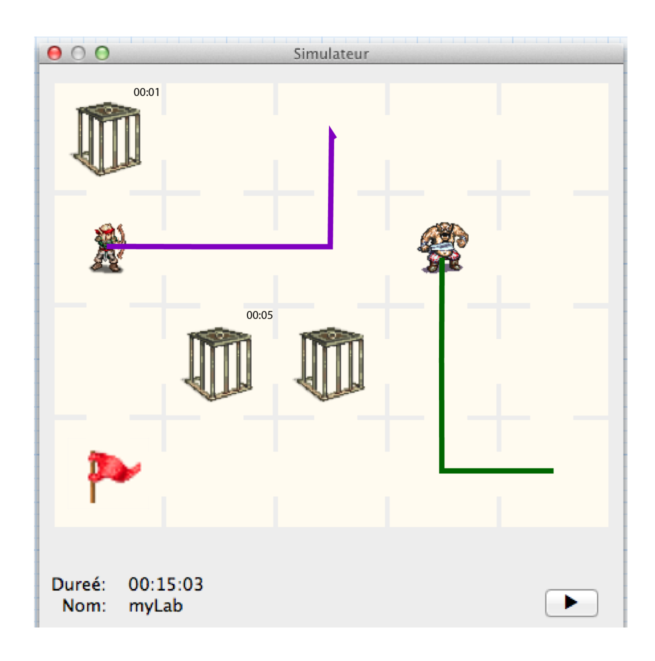


Figure 3 – Exemple d'écran du simulateur de labyrinthe (version initiale)

Objectif : Simuler un labyrinthe en forme d'une grille de taille N*M comprenant des salles et des personnages qui essayent de sortir du labyrinthe (définis dans l'atelier et l'éditeur du labyrinthe).

- Le simulateur contient une liste de tous les personnages ;
- Chaque personnage commence dans une salle définie par l'utilisateur ;
- Pour se déplacer, un personnage choisit la porte qu'il va chercher à ouvrir selon sa stratégie de mouvement (e.x. aléatoire). S'il arrive à ouvrir la porte, il se déplace vers la salle correspondante (cela signifie que la salle n'était pas bloquée). Si la salle correspondante est bloquée, le personnage essaye durant un certain temps (qui dépend de sa vitesse) d'ouvrir la porte. À la fin de ce temps (temps de mouvement), il peut essayer une autre salle ou éventuellement s'entêter à ouvrir cette porte. Le temps de mouvement d'un personnage dépend de sa vitesse : choisissez une constante (*MaxMovementTime*) pour tous les personnages, et calculez leur temps de mouvement en divisant MaxMovementTime par leur vitesse. Le personnage continue jusqu'à son arrivé à la sortie ;
- Si une salle devient bloquée quand un personnage est déjà dedans (pendent la durée de son temps de mouvement), il doit attendre jusqu'au moment ou la salle se débloque ;
- Un message est affiché (ex. console) chaque fois qu'un personnage arrive à la sortie.

Remarques :

- Chaque personnage est géré par son propre thread,
- Tous les personnages doivent partir en même temps,
- La vitesse de déplacement (temps de mouvement) vers la prochaine salle est affecté par leur vitesse,
- Chaque personnage a une stratégie de mouvement, mais pour le moment, tous les personnages se déplacent vers une des 4 salles voisines de façon aléatoire.
- Le programme termine lorsque tous les personnages arrivent à la sortie, ou l'utilisateur le termine.

Éditeur de labyrinthe
Version avancée à rendre pendant la dernière séance de TP-projet

L'éditeur de labyrinthe permet à l'utilisateur de :
- charger la description (propriétés) d'un labyrinthe existant,
- créer un nouveau labyrinthe en ajoutant les salles,
- ajouter une option permettant de « bloquer » quelques salles pour une certaine période *BlockTime* (permettant, périodique, ou aléatoire - voir version initial),
- **(new)** ajouter une option permettant de donner d'autres types des salles : des salles noires (sans visibilité), submergés (pleins d'eau) ...
- **(new)** ajouter des objets dans des salles particulières du labyrinthe parmi < clé, lampe de poche, jumelles, gilet de sauvetage, potion de soins >.

Fonctionnalités souhaitées :
- Sauvegarde et chargement du labyrinthe,
- Possibilité de spécifier si une salle peut devenir bloquée (donc on ne peut pas y accéder et si un personnage y est alors il est immobilisé).
- Définir la sortie du labyrinthe.
- **(new)** Possibilité de spécifier si une salle est noire ou submergée.
- **(new)** Possibilité d'ajouter un type d'objet dans une salle spécifique parmi < clé, lampe de poche, jumelles, gilet de sauvetage> et définir son quantité (e.x. combien des clés dans une salle).

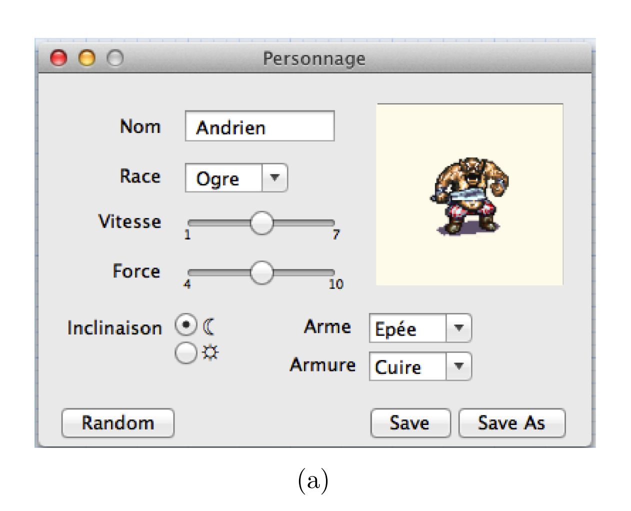
Objectif : Ajouter des variations des salles qui vont influer le parcours des personnages.
- **(new)** On a cinq types de salles : normales, bloquées, noires, submergées, et la sortie (unique), avec pour chacune un comportement diffèrent.
- Un personnage ne peut pas entrée dans une salle bloquée, et *passe son tour* (temps de mouvement) à essayer d'ouvrir la porte. Si par contre le personnage a une clé, il peut traverser la salle normalement, même les salles bloquées en permanence.
- **(new)** Si un personnage est dans une salle qui devient bloquée, il reste dedans jusqu'à qu'elle se débloque, sauf s'il a un clé.
- **(new)** Un personnage peut traverser une salle noir ou une salle submergée, mais de façon plus lent (2 et 3 fois son temps de mouvement normal respectivement). Par contre, si le personnage a une lampe de poche il peut traverser les salles noires à sa vitesse normale. De même s'il a un gilet de sauvetage pour les salles submergées.

Pour aller plus loin dans votre projet :
- l'éditeur de labyrinthe permet d'avoir d'autres types de salles avec leur difficulté et des objets possibles pour aider les personnages.
- les labyrinthes peuvent être plus que des grilles, par exemple des pavage hexagonal (donc une salle a 6 salles voisins), des vrais labyrinthes en méandres avec des impasses, etc.
- ...

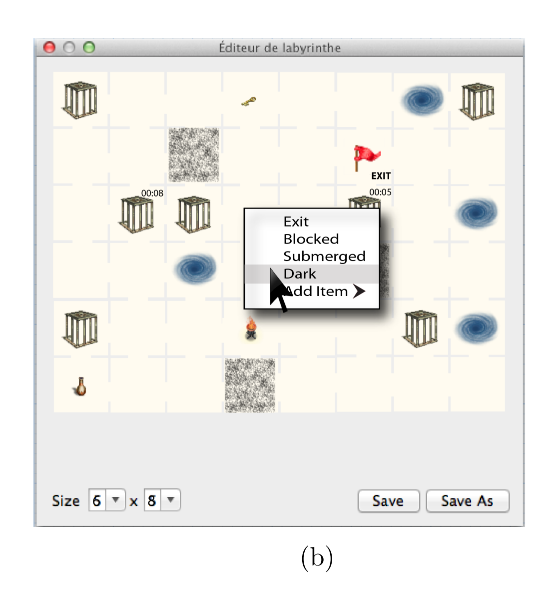
Atelier
Version avancée à rendre pendant la dernière séance de TP-projet

L'atelier permet à l'utilisateur de construire des personnages plus avancés :
- charger la description (propriétés) d'un personnage existant,
- créer un nouveau personnage en modifiant les propriétés d'un personnage existant,
- lui donner les propriétés de la version initiale, et en plus :
- (new) ajouter une inclinaison,
- (new) avoir des stratégies de mouvement plus élaborés,
- (new) posséder un sac d'objets pour faciliter sa progression dans le labyrinthe,
- (new) l'équiper pour le combat.
- (new) Finalement, avoir des personnages contrôlés par l'utilisateur.

Fonctionnalités souhaitées :
- Création, sauvegarde et chargement personnage.
- Création par l'utilisateur, aléatoire, ou en modifiant un personnage existant ("save as").



(a)



(b)

FIGURE 4 – Atelier et Éditeur. Exemples de fenêtres (a) de l'atelier pour la génération de nouveaux personnages et (b) de l'éditeur de labyrinthe avec des salles et des objets (versions avancées).

Objectif : Permettre la création d'un personnage à partir de zéro, aléatoirement ou à partir d'une autre.

Chaque personnage est défini par :
- son nom,
- sa race : ogre, humain, ou elfe,
- sa vitesse de mouvement : un entier entre 1-7 pour les ogres, entre 4-10 pour les humains, et 8-10 pour les elfes,
- sa force : un entier entre 4-10 pour les ogres, entre 1-7 pour les humains, et 1-3 pour les elfes,
- (new) son inclinaison, bon ou mauvais,
- (new) la taille de son sac pour collecter des objets (entre 2-4),
- (new) sa stratégie de mouvement,
- (new) son armure et son arme,
- (new) une représentation visuelle, par exemple une image ou une forme géométrique, avec des infos sur son inclinaison et son armure/arme.

La stratégie de mouvement :

- Un personnage contrôlé par l'utilisateur n'a pas une stratégie de mouvement, c'est l'utilisateur qui décide sa direction de déplacement.
- Un personnage autonome choisi de se déplacer vers une des 4 salles voisines de façon aléatoire. Si la salle est bloquée il doit attendre son temps de mouvement avant de ré-essayer (la même ou une autre salle).
- **(new)** En général, un personnage ne connais pas le contenu des salles autour (si elles sont déjà occupées ou le type des salles). Sauf s'il a des jumelles dans son sac. Dans ce cas il peu avoir une stratégie plus avancée et décider quelle salle est la plus rapide à traverser (pas occupée ou occupée par des alliés) et éviter des salles difficiles (occupées par des ennemies, bloquées et noires, ou submergées s'il n'a pas de lampe de poche ou de gilet de sauvetage).
- **(new)** Quand un personnage entre dans une nouvelle salle qui est occupée par un autre personnage il y a 4 possibilités :
  - Si la salle est noire il progresse normalement comme ils ne peuvent pas se voir (mais de façon plus lente, sauf s'il a une lampe) ;
  - Si le personnage qui occupe la salle est de la même race et inclinaison ils peuvent la partager ;
  - Si le personnage est d'une autre race mais de la même inclinaison, il doit revenir en arrière dans la salle précédente ;
  - Si le personnage est d'une autre inclinaison, il doit la combattre.

Combat : Chaque personnage qui entre en combat a un chance de succès qui est une combinaison de son attaque et de sa défense, ainsi qu'un facteur de chance :

- attaque = arme*force + chance (calculée pendant chaque combat, voir simulateur),
- défense = armure*vitesse + chance (calculée pendant chaque combat, voir simulateur)
- les différents objets ont des coefficients différents.
  - Par exemple un arc peut avoir un coef de 1.1, une épée de 1.2, une hache de 1.3.
  - Une armure en cuir peut avoir un coef de 1.1, une armure de cotte de mailles de 1.3, etc.
- Si un personnage est vaincu, il doit rester dans la salle 3 fois son temps de mouvement (le temps de guérir) avant d'essayer d'entrer dans une autre salle. Sauf s'il a une potion de soins qu'il peut boire et continuer normalement.

Sac : Chaque personnage a un sac avec une capacité pré-défini. S'il entre dans une salle avec un objet il doit l'ajouter dans son sac s'il y a de l'espace, la capacité du sac diminue alors d'une unité. Un objet dans le sac peut être utilisé une seule fois, ce qui libère une unité de capacité dans le sac. La seule façon de libérer de l'espace dans le sac est d'utiliser un de ses objet dans une salle correspondante.

Pour aller plus loin :

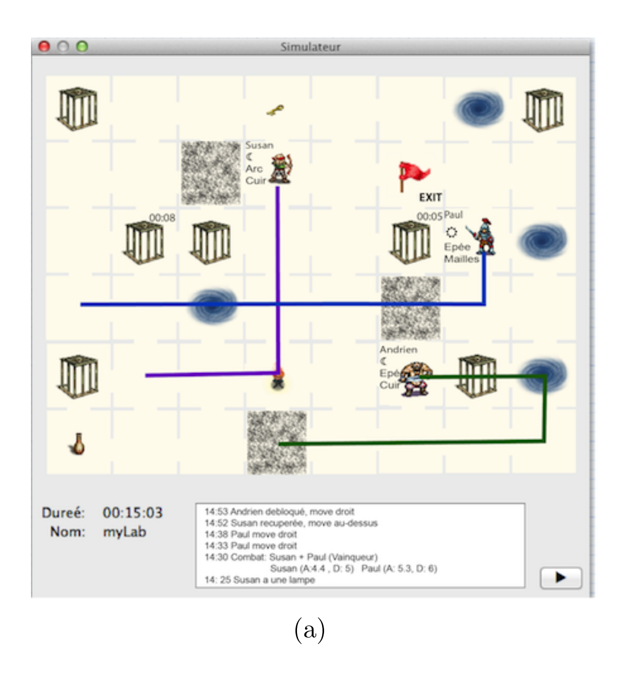
- Vous pouvez avoir d'autres stratégies de mouvement. Par exemple vous pouvez ajouter un objet "carte" qui montre ou est la sortie. Les personnages avec un carte peuvent essayer de choisir les salles qui minimise sa distance euclidien de la sortie ou la difficulté du chemin à emprunter.
- Vous pouvez avoir d'autres stratégies de collection d'objets. Par exemple vous pouvez avoir des personnages qui collectent que des clés ou des personnages qui ont des priorités et peuvent choisir de remplacer des items dans leur sacs avec des nouveaux items trouvés, mêmes pas utilisés.
- Lorsqu'un personnage est battu, ses objets (sauf la potion) reviennent au vainqueur.
- ...

Simulateur
Version avancée à rendre pendant la dernière séance de TP-projet

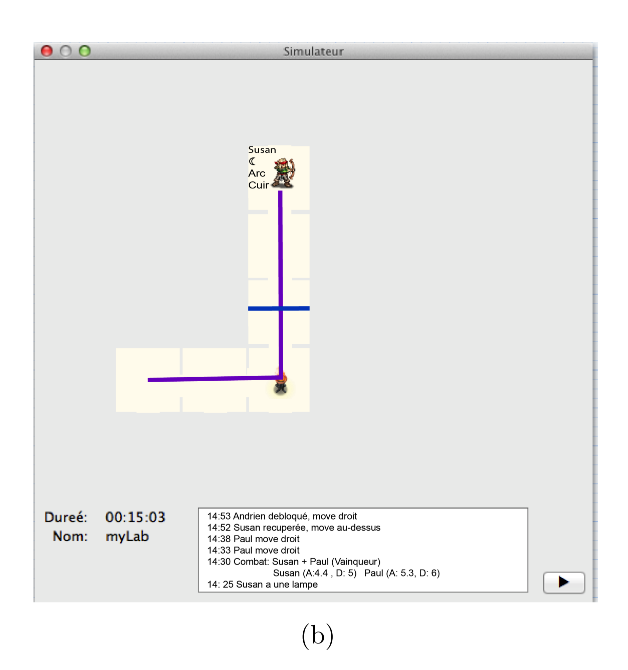
Le simulateur permet à l'utilisateur de visualiser les positions des personnages avancés et leur déplacements, et les items dans le labyrinthe. De plus il permet à l'utilisateur de contrôlés certains des personnages.

Fonctionnalités souhaitées :

- Chargement de personnages créés et sauvegardés dans un fichiers au travers de l'atelier ;
- Chargement d'un labyrinthe à partir du fichier créé dans l'éditeur de labyrinthe ;
- Visualisation du labyrinthe (salles et déplacements) ;
- **(new)** Sauvegarde de tous les événements et déplacements dans un fichier de log pour trouver les erreurs de programmation plus facilement ;
- **(new)** Gestions des combats ;
- **(new)** Gestions des quantités d'objets placés dans le labyrinthe ;
- **(new)** Gérer les événements d'utilisateur pour contrôler un personnage particulier ;
- **(new)** Avoir une visualisation particulière de la simulation quand il y a un personnage contrôlé par l'utilisateur ("fog of war").



(a)



(b)

Figure 5 – Exemples de fenêtres du simulateur du labyrinthe (version avancée), (a) normale et (b) fog of war (un personnage contrôlée par l'utilisateur).

Objectif : Simuler un labyrinthe en forme d'une grille de taille N*M comprenant des salles et des personnages qui essayent de sortir du labyrinthe (définis dans l'atelier et l'éditeur du labyrinthe).

- Le simulateur contient une liste de tous les personnages et objets (et leur quantités) ;
- Chaque personnage commence à une salle définie par l'utilisateur ;
- Un message est affiché (ex. console) chaque fois qu'un personnage arrive à la sortie ;
- Chaque personnage avance dans une des salles autour, en utilisant sa stratégie de mouvement (qui peut être aléatoire). Le temps pour avancer dans la salle suivante (temps de mouvement) dépend de sa vitesse, du type de la salle, et des objets qu'il a à sa disposition. Choisissez une constante (*MaxMovementTime*) pour tous les personnages, et calculez leur temps de mouvement normale en divisant MaxMovementTime par leur vitesse. Adaptez ce temps par

rapport au type de salle (voir Atelier et Éditeur). Le personnage progresse jusqu'à son arrivé à la sortie ;
- **(new)** Le simulateur calcule la quantité des objets restant dans les différentes salles du labyrinthe (les personnages peuvent collecter des objets) ;
- **(new)** Le simulateur calcule le résultats des combats entre deux personnages dans les salles (voir Remarques pour le calcul). Le vainqueur progresse normalement. Le perdant doit rester dans la salle pour se guérir (voir Atelier) sauf s'il a un potion.
- **(new)** S'il y a un combat dans une salle, les personnages qui essayent d'y entrer doivent attendre la fin du combat (la salle est inaccessible même pour les personnages avec un clé) ;
- Si un personnage est contrôlé par l'utilisateur, le simulateur est dans la visualisation "fog of war". C.à.d. la simulation tourne, mais les seules salles qui sont visibles sont les salles que le personnage de l'utilisateur a déjà traversé. Exceptionnellement, si le personnage de l'utilisateur a des jumelles, il peut aussi voir les 4 salles adjacentes.
- **(new)** L'utilisateur peut donner des instructions de mouvement et de collection d'objets en utilisant le clavier ou la souris.

Remarques :
- Chaque personnage est géré par son propre *Thread*,
- Tous les personnages doivent partir en même temps,
- Le programme termine lorsque tous les personnages arrivent à la sortie, ou que l'utilisateur le termine,
- **(new)** La vitesse de déplacement (temps de mouvement) vers la prochaine salle est affectée par leur vitesse, le type de la salle et leurs objets,
- **(new)** Chaque personnage a une stratégie de mouvement (voir Atelier), et peut entrer dans une salle ou non si la salle est déjà occupée (voir Atelier),
- **(new)** Quand deux personnages entre dans une salle il peuvent se combattre (voir Atelier pour les conditions de combat qui dépend en leur inclinaison).
  - Pour chaque personnage le simulateur calcule son attaque et sa défense, ainsi que 2 entiers aléatoires entre 0-5 (chance) qui sont ajoutés aux attaques et aux défenses.
  - Le résultat d'un combat est $Resultat = (attaque_1 - defense_2) - (attaque_2 - defense_1)$.
  - Si le *Résultat* est positif, le personnage 1 est le vainqueur, s'il est négatif, le personnage 2 est le vainqueur. Si le Résultat est 0 c'est un match nul et les deux personnages sont vainqueurs.

Pour aller plus loin :
- les personnages peuvent laisser une trace de leur trajet,
- contrôler d'autres paramètres pendant la simulation de façon dynamique (ex. le MaxMovementTime, ajout de salles particulières), sans arrêter la simulation,
- montrer le mouvement des personnage de façon avancée (animations),
- définir un trajet complexe pendant la simulation (plusieurs salles sélectionnées ensemble) pour contrôler le mouvement d'un personnage de l'utilisateur.
- ...

Conseils pour réaliser le projet

Des parties des programmes demandées sont relativement indépendantes, vous pouvez donc vous partager le travail (mais ce n'est pas une obligation). Voici quelques conseils :

1. **commencez simplement :** inutile de vouloir directement faire les versions avancées de chaque programme. Ces versions seront beaucoup plus simples à envisager lorsque vous aurez terminé la première partie du projet.
2. **décomposez vos problèmes :** concentrez vous sur un sujet précis plutôt que de tenter de réaliser les trois programmes en parallèle.
3. **faites le lien avec ce que vous savez déjà fait :** les notions et programmes que vous avez fait en TP sont suffisantes pour construire vos premières versions de programme.

Remise des documents et du programme

Pour la remise des documents, vous devez remettre en mains propres un **document papier** lors des séances de TP ayant lieu à la date donnée. En *haut de la première page*, n'oubliez pas de mettre vos **noms**, **prénoms**, la mention **"Développement Logiciel : projet"** et l'intitulé du document (ex. manuel d'utilisation). Inutile de prendre une page entière pour ces informations. **Agraffez** (ou reliez) vos documents.

Pour la remise de votre programme, on vous demande d'envoyer aux **deux** enseignants (chargé de cours et votre chargé de TP), une archive au format **zip** ou **tgz** de vos sources. Vous devez envoyer cette archive lors de la dernière séance.

Attention : Testez votre archive. Vous pourrez consulter le site du cours pour voir si nous avons reçu (ou non) votre archive.

Contacts pour l'envoi des documents et des sources de votre projet

Chargés de TP :
Jérémie Garcia,
Driss Sadoun

Chargée de cours :
Anastasia Bezerianos

Merci d'envoyer tous les documents à votre chargé de TP uniquement, en mettant en copie votre chargée de cours.

=> n'oubliez pas de mettre "[DevLog]" au début du sujet de votre mail (pour éviter d'être classer en spam). Par exemple : "[DevLog] Question sur ...". Nous pourrons vous donner des indications générales mais en aucun cas résoudre des problèmes de programmation.